



UNIVERSITÀ DEGLI STUDI DI PALERMO

Framework Grafico per raspberrypi 3 e 4 attraverso
l'utilizzo del protocollo Mailbox

Prof. Daniele Peri

Gatto Luigi

Anno Accademico 2019-2020

Sommario

1. Introduzione
2. Sistema hardware
3. Descrizione del sistema
4. Modello in uso e configurazione
5. Codice Forth ed esempi
6. Future implementazioni

1. Introduzione

Il sistema progettato e sviluppato ha come scopo principale quello di semplificare l'utilizzo del framebuffer tramite l'ausilio del 'protocollo' mailboxes.

Oltre agli esempi dimostrativi, infatti, vi è la possibilità di consultare tutte le words create in modo tale da poter imparare correttamente ed in un modo semplice come poter effettuare elaborazioni grafiche su un raspberry pi 4 con SO piJFORTHOS (anche nel raspberry PI 3 tramite opportune modifiche, ovvero modificare la variabile RASBP4 da FE000000 a 3F000000 e anche gli indirizzi del timer.).

```
( COSTANTI )  
  \nel rasp3 è 3F000000  
  FE000000 CONSTANT RASBP4
```

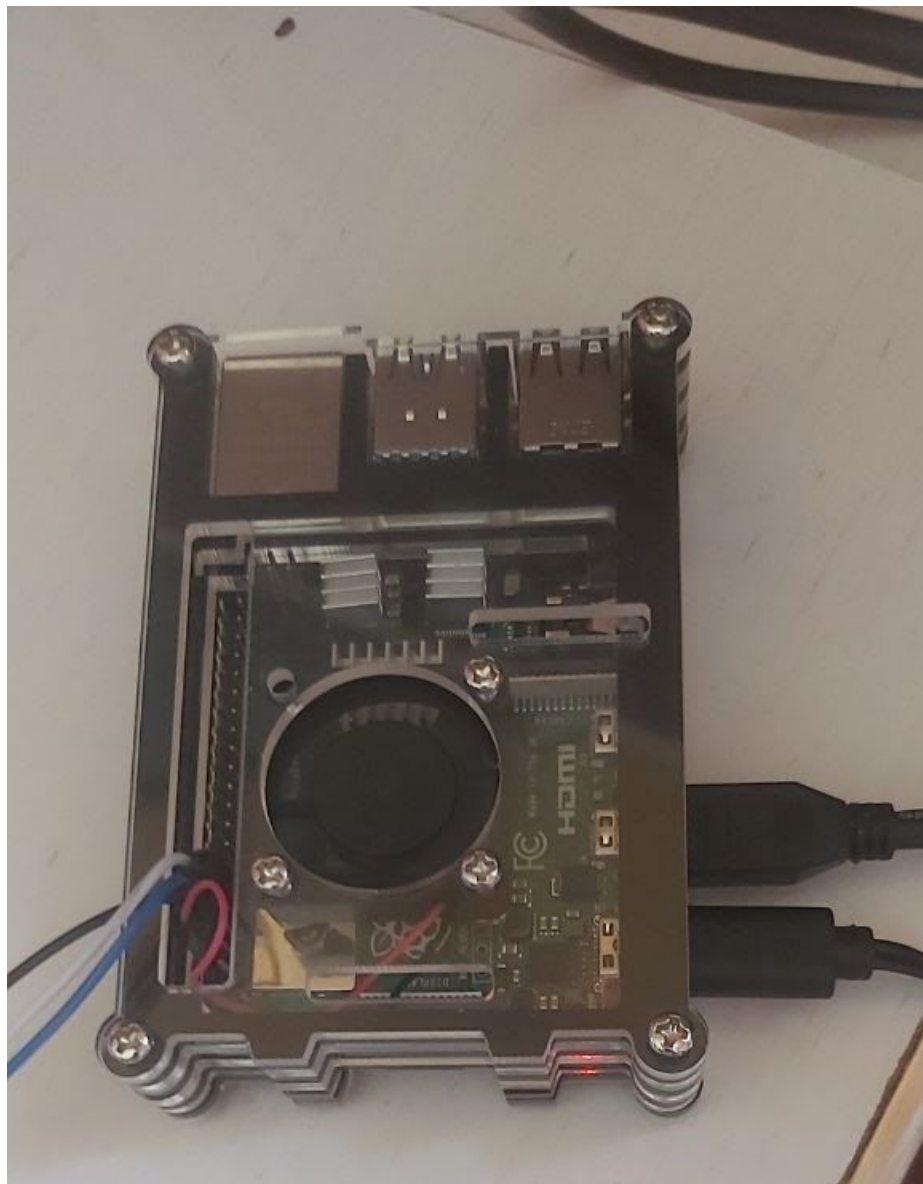
Per questo motivo, infatti, è fortemente consigliato consultare questo progetto non come un framework completo e fine a se stesso ma come linea guida per progetti futuri o semplici esperimenti da fare con il proprio raspberry.

L'intero codice forth è commentato e le spiegazioni sintattico-logiche sono descritte all'interno di questo documento e del codice stesso.

2. Sistema Hardware

Per lo sviluppo del progetto sono state utilizzati i seguenti hardware e strumenti:

- **Computer:**
E' necessario un generico computer con un SO in grado di supportare un software che simuli un terminale (e.s Putty o Zoc o picocom).
- **Raspberry:**
Il progetto è basato su un Raspberry PI 4 4GB, ma cambiando correttamente un'indirizzo, specificato in seguito, è possibile utilizzare tutte le funzioni su un raspberry PI 3.
- **Scheda SD**
All'interno della scheda SD vi deve essere caricato pijFORTHos, Un sistema operativo bare metal per Raspberry Pi, basato su Jonesforth-ARM.
<https://github.com/organix/pijFORTHos>
- **UART**
L' Universal Asynchronous Receiver-Transmitter è un dispositivo hardware che permette la trasmissione di dati tra un computer ed il raspberry.
è stato utilizzato un FTDI FT232RL usb-ttl
www.amazon.it/FT232RL
- **Monitor**
Un monitor di risoluzione qualunque, l'importante è che abbia come ingresso HDMI.
- **Cavi**
Cavo alimentazione raspberry.
Cavo HDMI-microHDMI per collegare il monitor al raspberry
Cavo per collegare FT232 al raspberry



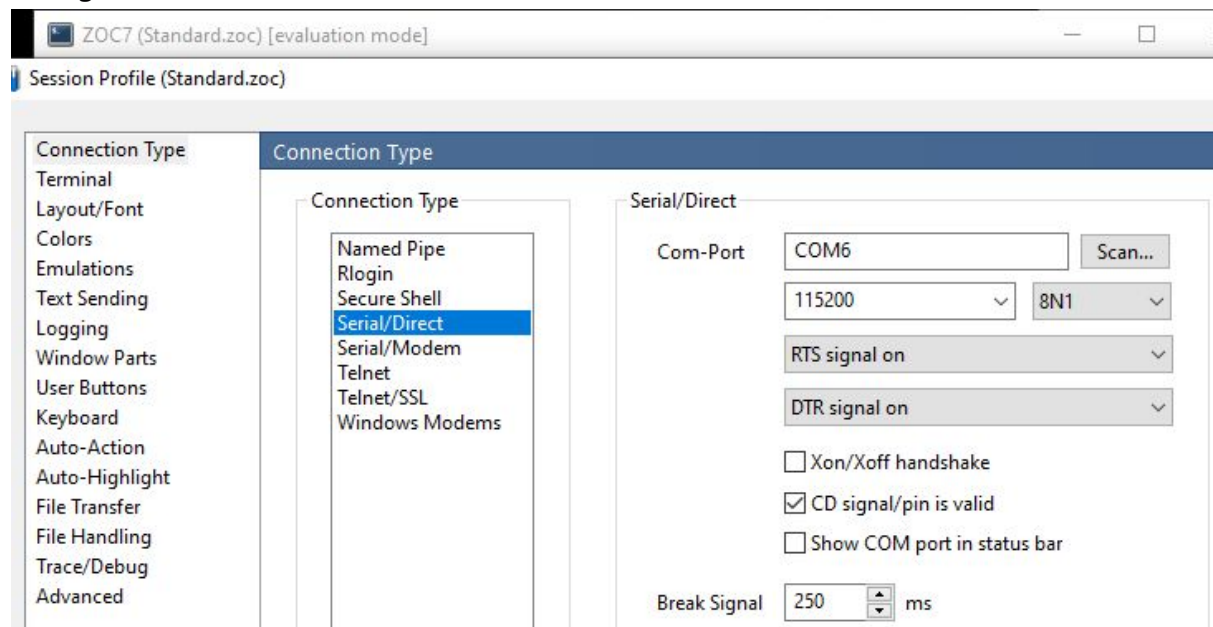
3. Descrizione del sistema

In seguito saranno spiegati in modo approfondito tutte le componenti del sistema.

3.1. pijFORTHos è composto da:

- 3.1.1. bootcode.bin → bootloader
- 3.1.2. fixup.dat → Linker file per il raspberry
- 3.1.3. kernel7.img → Immagine dell'interprete FORTH modificato
- 3.1.4. start.elf → Firmware di base
- 3.1.5. jonesforth.f → per implementazioni base
- 3.1.6. Complete.f → dizionario forth completo.
- 3.1.7. CompleteLight.f → dizionario essenziale.

3.2. Configurazione ZOC7



il numero della porta, nel mio caso 6, può cambiare ed è ricavabile tramite "gestione dispositivi" in windows.

configurazione standard:

baudrate: 115200

bits: 8

parity : N

stopbits: 1

input delay 1 ms per facilitare il 'copia ed incolla' nel terminale.

3.3. Framebuffer

Il framebuffer è una zona di memoria condivisa tra ARM ed il VC in cui vengono memorizzate le informazioni destinate all'output per la rappresentazione di un intero fotogramma (chiamato frame) sullo schermo.

Nel framebuffer sono contenute le informazioni riguardanti il colore di ciascun pixel, ovvero di ciascun punto dello schermo.

Queste possono essere a 1-bit (bianco e nero), 4-bit o 8-bit (modalità a palette), 16-bit o 24 bit (highcolor/ truecolor).

Le dimensioni del framebuffer dipendono dalla risoluzione dell'output video e dai bit necessari per rappresentare il colore di ciascun pixel.

Nella computer grafica, la palette è un insieme dei colori disponibili in un programma di grafica o di disegno, ogni colore della palette è associato ad un indice che identifica il colore nella palette stessa.

La rappresentazione dei pixel di un'immagine attraverso indici nella palette permette un drastico risparmio di memoria ed ha una buona efficienza in termini di costo computazionale ma ha come difetto un limitato range di colori disponibili.

3.4. Mailbox

Tutte le informazioni sulla mailbox e le immagini sono state prese da:

<https://github.com/raspberrypi/firmware/wiki/Mailboxes>.

Le Mailboxes (MB) facilitano la comunicazione tra l'ARM ed il VideoCore (VC).

Ogni mailbox è una coda FIFO 8-deep (con capacità 8) di parole di 32 bit, che può essere letta/scritta dall'ARM e dal VC.

Solo lo stato della mailbox 0 può innescare un interrupt sull'ARM, quindi la MB 0 è sempre utilizzata per comunicazioni dal VC all'ARM, mentre la MB 1, al contrario, è utilizzata per comunicazioni dall'ARM al VC. L'ARM non dovrebbe mai essere in grado di scrivere nella MB 0 o leggere dalla MB 1.

Il buffer di risposta e di richiesta è lo stesso, quindi quando viene effettuata una richiesta la risposta sovrascrive tutte le informazioni relative alla richiesta.

Questo permette un risparmio di spazio ma non solo:

Essa permette potenzialmente di effettuare più richieste in modo asincrono, un utente può infatti inviare più richieste in buffer differenti e non è necessario creare dinamicamente altri buffer di risposta dato che gli stessi buffer di richiesta verranno sovrascritti.

Il 'protocollo' mailbox garantisce una sorta di sicurezza in caso di richiesta errata, infatti se la richiesta è sintatticamente scorretta o troppo lunga non verrà settata alcun tipo di risposta.

3.4.1. Canali

- 0: Power management
- 1: Framebuffer
- 2: Virtual UART
- 3: VCHIQ
- 4: LEDs
- 5: Buttons
- 6: Touch screen
- 7: Non specificato
- 8: Property tags (ARM -> VC)
- 9: Property tags (VC -> ARM)

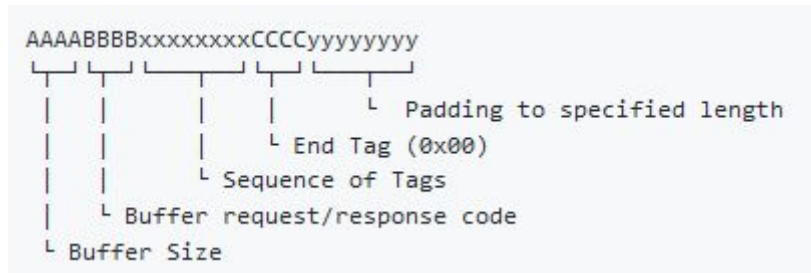
3.4.2. Registri mailbox

La tabella seguente mostra gli offset del registro per le diverse mailboxes:

Mailbox	Read/Write	Peek	Sender	Status	Config
0	0x00	0x10	0x14	0x18	0x1c
1	0x20	0x30	0x34	0x38	0x3c

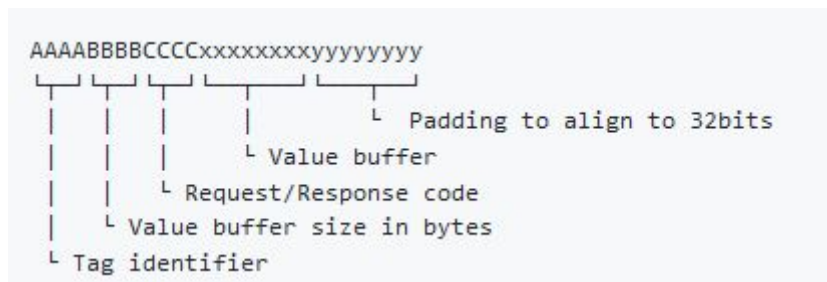
3.4.3. Messaggio

Il contenuto del messaggio può essere descritto graficamente nel seguente modo:



- **Buffer size**: valore da 32 bit che specifica la dimensione del buffer, questa include la dimensione dell'intestazione, l'end tag e il padding.
- **Buffer request/response code**: valore da 32 bit, nel caso di una richiesta deve essere 0x0 (tutti gli altri valori sono riservati), invece, nel caso di una risposta è pari a 0x80000000 in caso di richiesta completata con successo e 0x80000001 altrimenti (tutti gli altri valori sono riservati).
- **Sequence of tags**: contiene tutte le richieste concatenate da effettuare alla mailbox (la struttura di queste sono spiegate più dettagliatamente nel paragrafo successivo)
- **End tag**: valore da 32 bit, deve essere pari a 0x0
- **Padding**: campo per allineare il buffer a 32 bit nel caso in cui qualche cella precedente sia da 32 bit.

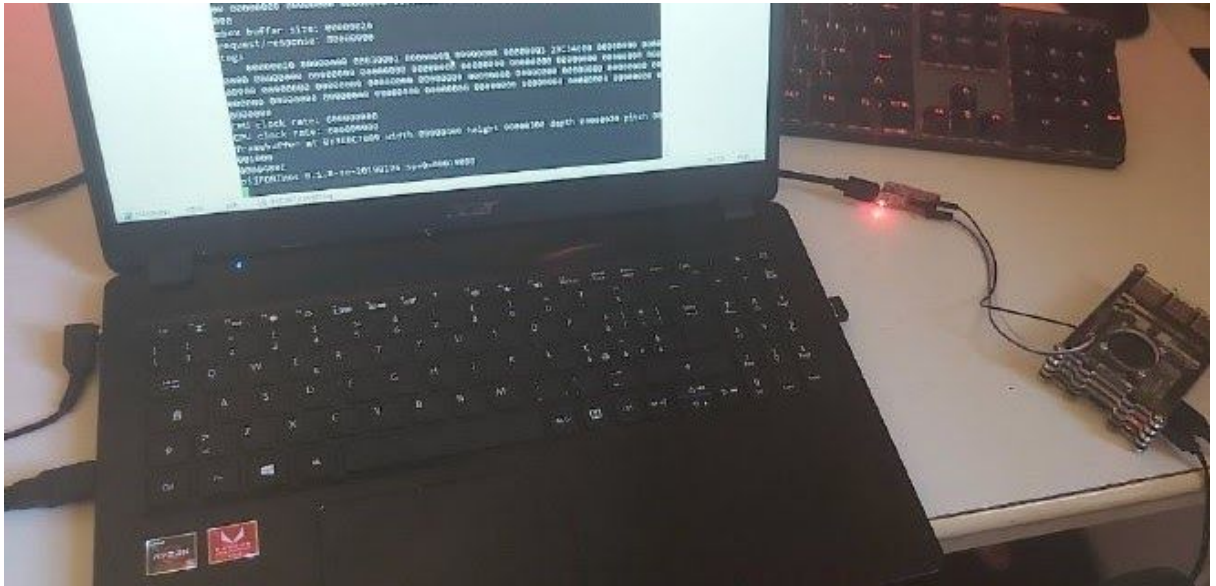
3.4.4. Tag format



- Tag identifier: valore da 32 bit che specifica la richiesta (ad ogni richiesta è associato un tag)
- Value buffer size in bytes: valore da 32 bit che indica la dimensione del buffer in bytes
- Request/response code: valore da 32 bit. Per una richiesta deve avere il bit meno significativo settato a 0 e i restanti bit sono riservati, invece, in caso di risposta il bit meno significativo è settato a 1 e i restanti bit indicano la dimensione dei valori in bytes.
- Value buffer: contiene i valori per ogni tag
- Padding: per allineare il tag a 32 bit nel caso in cui qualche cella precedente sia da 32 bit.

4. Modello in uso e configurazione

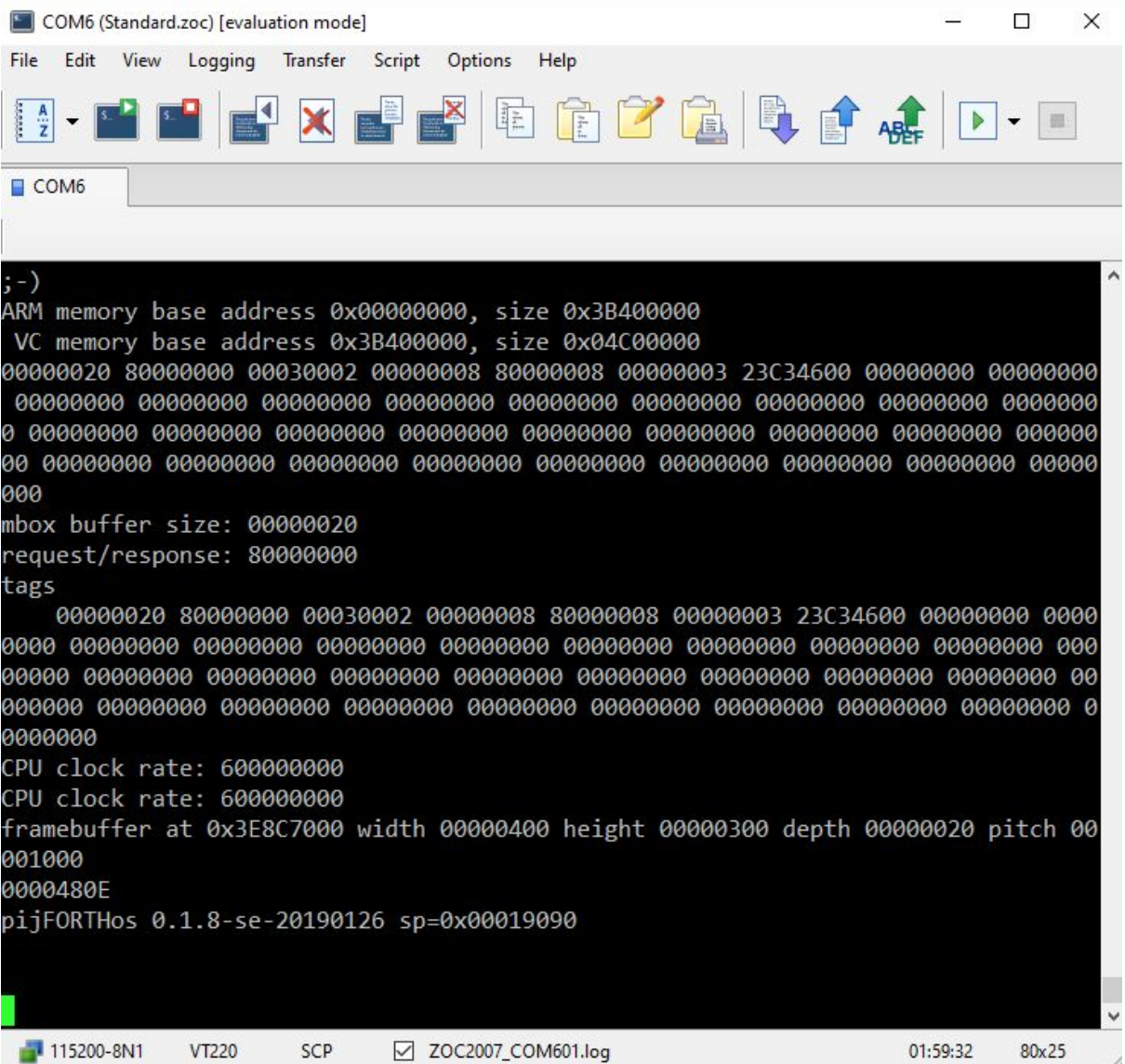
E' possibile raffigurare l'intero sistema come in figura:



L'output del raspberry viene trasmesso al monitor tramite l'uscita HDMI.
Il raspberry è collegato al PC tramite l'UART FT232RL.

Una volta avviato il raspberry, se la configurazione di `pijFORTHos` è corretta ed il software che simula il terminale è correttamente configurato, apparirà una faccina nella schermata del terminale per far capire che il sistema è pronto per essere avviato.

Una volta premuto "enter" nel computer, verranno mostrate a schermo molte informazioni di configurazione:



The screenshot shows a terminal window titled "COM6 (Standard.zoc) [evaluation mode]". The window has a menu bar with "File", "Edit", "View", "Logging", "Transfer", "Script", "Options", and "Help". Below the menu bar is a toolbar with various icons. The main area of the window displays the following text:

```
; -)
ARM memory base address 0x00000000, size 0x3B400000
VC memory base address 0x3B400000, size 0x04C00000
00000020 80000000 00030002 00000008 80000008 00000003 23C34600 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000
mbox buffer size: 00000020
request/response: 80000000
tags
    00000020 80000000 00030002 00000008 80000008 00000003 23C34600 00000000 00000000 00000000
0000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000
CPU clock rate: 600000000
CPU clock rate: 600000000
framebuffer at 0x3E8C7000 width 00000400 height 00000300 depth 00000020 pitch 00000100
0000480E
pijFORTHos 0.1.8-se-20190126 sp=0x00019090
```

The status bar at the bottom of the window shows "115200-8N1", "VT220", "SCP", a checked box next to "ZOC2007_COM601.log", "01:59:32", and "80x25".

cpu clock rate, indirizzo di memoria base di ARM e VC ed il framebuffer " di base " pre-configurato del pijFORTHos. Esso è utilizzabile soltanto se è collegato un cavo micro-hdmi ad uno schermo HDMI al primo ingresso (quello vicino all'alimentazione).

N.B Per chi utilizza ZOC7 è possibile utilizzare un REXX pre-impostato all'interno della cartella del progetto. (GATTOMAILBOX.zrx)

4.1. Mailbox

Le operazioni effettuate tramite la mailbox sono:

- Inizializzazione framebuffer.
- Inizializzazione palette.
- Inizializzazione words per blank-screen e release-buffer.
- Inizializzazione words per il get di informazioni.

Tutti i tag utilizzati sono stati presi da:

https://github.com/Mailbox_Tag

Parola utilizzata per inviare la richiesta:

```
( Wippa tutta la mailbox )
: WIPE ( -- )

    BEGIN
        0 MAILBOX @
        DROP
        1 MAILBOX STATUS @
        MAILBOXEMPTY =
    UNTIL
;

( Scrittura in mailbox previo wipp )
: WRITE ( addr -- )

    WIPE
    MAILBOXCHANNEL OR
    1 MAILBOX WR !
;
```

E' possibile implementare ulteriormente altre parole utilizzando la seguente sintassi.

SET

Set virtual (buffer) width/height

- Tag: 0x00048004
- Request:
 - Length: 8
 - Value:
 - u32: width in pixels
 - u32: height in pixels
- Response:
 - Length: 8
 - Value:
 - u32: width in pixels
 - u32: height in pixels

The response may not be the same as the request so it must be checked. May be the previous width/height or 0 for unsupported.

```
: SETVIRTUAL
    SETVIRTUALWH BUFFER 7 cell !
    8 BUFFER 8 cell !
    STARTMAILBOX BUFFER 9 cell !
    WIDTH BUFFER A cell !
    HEIGHT BUFFER B cell !
;
```

START_MAILBOX (0, non presente poiché dichiarata prima) → Tag (SET_VIRTUAL_WH = 48004) → dimensione della richiesta (8 length richiesta) → START MAILBOX → values → END_MAILBOX (0, non presente poiché dichiarato dopo).

Blank screen

- Tag: 0x00040002
- Request:
 - Length: 4
 - Value:
 - u32: state
 - State:
 - Bit 0: 0=off, 1=on
 - Bits 1-31: reserved for future use (set to 0)
- Response:
 - Length: 4
 - Value:
 - u32: state
 - State:
 - Bit 0: 0=off, 1=on
 - Bits 1-31: reserved for future use

```
: BLANKSCREENY
    30 BUFFER 0 CELL !
    STARTMAILBOX BUFFER 1 CELL !
    BLANKSCREEN BUFFER 2 CELL !
    4 BUFFER 3 CELL !
    STARTMAILBOX BUFFER 4 CELL !
    1 buffer 5 cell !
    ENDMAILBOX BUFFER 6 CELL !

    BUFFER WRITE
;

: BLANKSCREENN
    30 BUFFER 0 CELL !
    STARTMAILBOX BUFFER 1 CELL !
    BLANKSCREEN BUFFER 2 CELL !
    4 BUFFER 3 CELL !
    STARTMAILBOX BUFFER 4 CELL !
    0 buffer 5 cell !
    ENDMAILBOX BUFFER 6 CELL !

    BUFFER WRITE
;
```


GET

Get virtual (buffer) width/height

Note that the "virtual (buffer)" size is the portion of buffer that is sent to the display device, not the resolution the buffer itself. This may be smaller than the allocated buffer size in order to implement panning.

- Tag: 0x00040004
- Request:
 - Length: 0
- Response:
 - Length: 8
 - Value:
 - u32: width in pixels
 - u32: height in pixels

```
: GETVIRTUALWHFRAMEBUFFER  
  
30 BUFFER 0 CELL !  
STARTMAILBOX BUFFER 1 CELL !  
GETVIRTUALWH BUFFER 2 CELL !  
ENDMAILBOX BUFFER 3 CELL !  
BUFFER WRITE  
;
```

INIZIALIZZO UN BUFFER → START_MAILBOX → TAG →
END_MAILBOX → BUFFER SCRIVI.

La risposta verrà sovrascritta nello stesso buffer della richiesta, come spiegato nel paragrafo precedente.

4.1.1. Configurazione framebuffer

Le impostazioni nel framebuffer sono:

Allocate buffer

- Tag: 0x00040001
- Request:
 - Length: 4
 - Value:
 - u32: alignment in bytes
- Response:
 - Length: 8
 - Value:
 - u32: frame buffer base address in bytes
 - u32: frame buffer size in bytes

If the requested alignment is unsupported then the current base and size (which may be 0 if not allocated) is returned and no change occurs.

```
: ALLOCATEFRAMEBUFFER ( -- )
  STARTMAILBOX BUFFER 1 cell !

  SETPHYSICAL
  SETVIRTUAL
  SETDEPTH

  ALLOCATEBUFFER BUFFER 10 cell !
  8 BUFFER 11 cell !
  8 BUFFER 12 cell !
  1000 BUFFER 13 cell !

  ENDMAILBOX BUFFER 14 cell !

  BUFFER WRITE
;
```

Vi è la possibilità di impostare arbitrariamente i valori di configurazione del framebuffer anche se bisogna osservare alcuni accorgimenti:

- La profondità dei colori può essere o 4 oppure 8 (per utilizzare la palette),
- La width e la height possono essere scelti arbitrariamente ma inferiore rispetto alla dimensione dello schermo se si vogliono evitare artefatti grafici.

Il settaggio di nella configurazione è 400x300 per puro scopo dimostrativo.

```
( PROPRIETA )
8 CONSTANT BITSPIXEL ( profondità del framebuffer attuale )
400 CONSTANT WIDTH ( altezza framebuffer )
300 CONSTANT HEIGHT ( larghezza framebuffer )
```

4.1.2. Configurazioni delle palette

Set palette

- Tag: 0x0004800b
- Request:
 - Length: 24..1032
 - Value:
 - u32: offset: first palette index to set (0-255)
 - u32: length: number of palette entries to set (1-256)
 - u32...: RGBA palette values (offset to offset+length-1)
- Response:
 - Length: 4
 - Value:
 - u32: 0=valid, 1=invalid

```
: INIZIALIZZAPALETTE ( -- )
  50 BUFFER 0 cell !
  STARTMAILBOX BUFFER 1 cell !
  SETPALETTE BUFFER 2 cell !
  E4 BUFFER 3 cell !
  STARTMAILBOX BUFFER 4 cell !
  PALETTEBACKGROUND BUFFER 5 cell !
  LASTPALETTE BUFFER 6 cell !

  ( PALETTEBACKGROUND )
  FF000000 BUFFER 7 cell !
  ( PALETTELINE )
  FFFFFFFF BUFFER 8 cell !
  ( PALETTELINE2 )
  ffaab0c3 BUFFER 9 cell !
  ( PALETTELINE3 )
  FFFFFFFF BUFFER 10 cell !

  ENDMAILBOX BUFFER 7 LASTPALETTE + cell !

  BUFFER WRITE
;
```

Se si sono capite le configurazioni precedenti, qui l'unica differenza è il colore che attribuisco ad ogni indice.
Potenzialmente potrei descrivere quante altre 'linee' colorate voglio continuando la sintassi:

PALETTE_LINE_X
COLORE BUFFER X cell !

e dichiarando le costanti:

```
( INDICI PALETTE 0 background 1 LINE 2 LINE2 3 LINE3)
0 CONSTANT PALETTEBACKGROUND
1 CONSTANT PALETTELINE
2 CONSTANT PALETTELINE2
3 CONSTANT PALETTELINE3

\ Nei vari forum viene sempre richieste un indice per indicare l'ultima palette, ho provato con un numero da 0 a A e da errore, ho provato con 1A e funziona
1A CONSTANT LASTPALETTE
```

X CONSTANT_LINE_X

5. Codice Forth ed esempi

In questo paragrafo verrà mostrato il codice forth per le implementazioni effettuate nel progetto come lo screensaver, la stampa di cuori, la creazione del tavolo, etc.

Iniziamo parlando del timer:

(TIMER)

```
( TIMER )
FE003000 constant CS
FE003004 constant CLO
FE003008 constant CHI
FE00300C constant C0
: delay CLO @ + C0 ! BEGIN CS @ 1 = UNTIL ;

: timer
20000 delay
;
```

File contenente le word per poter utilizzare le funzionalità del timer di sistema.

Le costanti sono:

- CS = 0x3F003000 → Registro di controllo / stato del System Timer
- CLO = 0x3F003004 → Registro contenente i 32 bit inferiori del System Timer Counter
- CHI = 0x3F003008 → Registro contenente i 32 bit superiori del System Timer Counter
- C0 = 0x3F003010 → Indirizzo del registro del System Timer Compare 0

Il timer è stato usato per temporizzare il sistema per passare da un colore di sfondo ad un altro.

E' possibile personalizzarlo levando il valore 20000 e richiamare la word passando come parametro un valore a propria scelta.

(*CURSORE*)

Il protocollo mailbox mette a disposizione un cursore a schermo customizzabile.

Il cursore e' stato implementato in modo tale da dare l'idea della posizione su schermo e semplificare così la scrittura utilizzando la famiglia di words DRAWXX.

Set Cursor Info

- Tag: 0x00008010
- Request:
 - Length: 24
 - Value:
 - u32: width
 - u32: height
 - u32: (unused)
 - u32: pointer to pixels
 - u32: hotspotX
 - u32: hotspotY
- Response:
 - Length: 4
 - Value:
 - u32: 0=valid, 1=invalid

Format is 32bpp (ARGB). Width and height should be ≥ 16 and $(width * height) \leq 64$.

```
: SETCURSORINFO
  50 BUFFER 0 CELL !
  STARTMAILBOX BUFFER 1 cell !
  CURSORINFO BUFFER 2 cell !
  24 buffer 3 CELL !
  STARTMAILBOX BUFFER 4 cell !
  4 buffer 5 cell !
  2 buffer 6 cell !
  0 BUFFER 7 CELL !
  2 buffer 8 cell !
  0 BUFFER 9 CELL !
  0 BUFFER A CELL !
  ENDMAILBOX buffer B cell !

  BUFFER WRITE
;
```

Set Cursor State

- Tag: 0x00008011
- Request:
 - Length: 16
 - Value:
 - u32: enable (1=visible, 0=invisible)
 - u32: x
 - u32: y
 - u32: flags
- Response:
 - Length: 4
 - Value:
 - u32: 0=valid, 1=invalid

The flags control: bit0: clean=display coords, set=framebuffer coords

```
: SETCURSOR
50 BUFFER 0 CELL !
STARTMAILBOX BUFFER 1 cell !
CURSORSTATE BUFFER 2 cell !
16 buffer 3 CELL !
STARTMAILBOX BUFFER 4 cell !
1 buffer 5 cell !
cursorx @ buffer 6 cell !
cursory @ buffer 7 cell !
1 buffer 8 cell !
ENDMAILBOX buffer 9 cell !

BUFFER WRITE

;
```

Utilizzando la famiglia delle parole DRAWXX, spiegata in seguito, è possibile effettuare veri e propri disegni utilizzando come riferimento il cursore che si sposterà automaticamente una volta effettuato il comando.

(SCREENSAVERS)

Il primo, semplice, esempio grafico di utilizzo delle potenzialità della mailbox è quello di creare dei screensaver. nel primo caso coloro il background di colori diversi ogni circa mezzo secondo.

```
( EXAMPLES )
: SCREENSAVERSTART
52 45 56 41 53 4E 45 45 52 43 53 emit emit emit emit emit emit emit emit emit emit
;

: SCREENSAVERFINISH
4F 54 49 4E 49 46 emit emit emit emit emit emit
;

: SCREENSAVER1

SCREENSAVERSTART
FFFFFFF CHANGEBACKGROUND
timer timer
FF00000 CHANGEBACKGROUND
timer timer
FFFF000 CHANGEBACKGROUND
timer timer
FF00FF0 CHANGEBACKGROUND
timer timer
FF000FF CHANGEBACKGROUND
timer timer
FFFFFFF CHANGEBACKGROUND
timer timer
FF00000 CHANGEBACKGROUND

SCREENSAVERFINISH
```

Il secondo, un pò piu complesso ma sempre di facile realizzazione:

```
: SCREENSAVER2
CLEAN
-90 12 movexy
FF0000ff CHANGELINE1
line1 SETDEFAULTLINE
30000 DRAWHR

-90 12 movexy
FFFF00ff CHANGELINE2
LINE2 SETDEFAULTLINE
30000 DRAWHR

-90 12 movexy
FF808080 CHANGELINE1
line1 SETDEFAULTLINE
30000 DRAWHR

-90 12 movexy
FF008080 CHANGELINE2
LINE2 SETDEFAULTLINE
30000 DRAWHR

-90 12 movexy
FFFFFFF CHANGELINE1
line1 SETDEFAULTLINE
30000 DRAWHR

CLEAN
```

permette di colorare l'intero schermo con colori diversi tramite una sorta di animazione.

Dato che viene colorato pixel per pixel, l'effetto finale " a finestra " è un buon modo per testare e vedere le potenzialità della mailbox.

(HEART)



Una figura un po più complessa, come un cuore, può essere facilmente scritta tramite i seguenti comandi:

```
: HEART
    200 200 movexy
    line1 5 5 obbul
    line1 4 5 obbd1
    line1 4 5 obbd1
    line1 5 10 obbdr
    line1 5 10 obbur
    line1 4 5 obbul
    line1 4 5 obbul
    line1 5 5 obbd1
    ff0000ff CHANGELINE1
;
```

Modificando la funzione a dovere, possiamo far diventare la funzione dinamica ed avere un risultato di questo tipo:



con una sintassi molto semplice:

```

: HEART ( posx posy -- )
    movexy
    line1 5 5 obbul
    line1 4 5 obbd1
    line1 4 5 obbd1
    line1 5 10 obbdr
    line1 5 10 obbur
    line1 4 5 obbul
    line1 4 5 obbul
    line1 5 5 obbd1
    ff0000ff CHANGELINE1
;

: HEARTS ( posx posy -- )
    50 50 HEART
    100 50 heart
    150 50 heart
    200 50 heart
    250 50 heart
    300 50 heart

    50 100 HEART
    100 100 heart
    150 100 heart
    200 100 heart
    250 100 heart
    300 100 heart

    50 200 HEART
    100 200 heart
    150 200 heart
    200 200 heart
    250 200 heart
    300 200 heart
;

```

N.B si ricorda che i valori delle posizioni sono in esadecimale, giusto il caso vuole che l'esempio mostra multipli di 50.

(TABLE)

Per la creazione del tavolo si è pensato di dividere in 'pezzi' l'oggetto. Possiamo identificare come foot (piedi del tavolo) e body (il 'sopra' del tavolo).

```
( TABLE EXAMPLE )
variable foot
variable body
: TAVOLO ( colorbody colorfoot -- )
foot !
body !

    foot @ 0A4 188 0B8 DRAWV \ PUNTA ALTA
    foot @ 0A0 188 0B8 DRAWV \ PUNTA ALTA
    foot @ 084 198 140 DRAWV \ PUNTA bassa
    body @ 088 198 140 DRAWV \ PUNTA bassa
    body @ 09c 180 10 DRAWH
    body @ 098 184 18 DRAWH
    body @ 094 188 20 DRAWH
    body @ 090 18c 28 DRAWH
    body @ 08c 190 30 DRAWH
    body @ 088 194 38 DRAWH
    body @ 084 198 40 DRAWH
    body @ 080 19C 48 DRAWH
    body @ 07C 1a0 50 DRAWH
    body @ 078 1a4 58 DRAWH
    body @ 074 1a8 60 DRAWH
    body @ 070 1ac 68 DRAWH
    body @ 06C 1b0 70 DRAWH
    body @ 068 1b4 78 DRAWH
    body @ 064 1b8 80 DRAWH
    body @ 060 1bc 88 DRAWH
    body @ 05C 1c0 8C DRAWH
    body @ 058 1C4 8C DRAWH
    body @ 054 1C8 8C DRAWH
    body @ 050 1CC 8C DRAWH
    body @ 04C 1d0 8C DRAWH
    body @ 048 1d4 8C DRAWH
    body @ 044 1d8 8C DRAWH

    foot @ 044 1d8 0AC DRAWV \ sinistra verticale
    foot @ 048 1d8 0AC DRAWV \ sinistra verticale
```

```
foot @ 044 1d8 0AC DRAWV \ sinistra verticale
foot @ 048 1d8 0AC DRAWV \ sinistra verticale

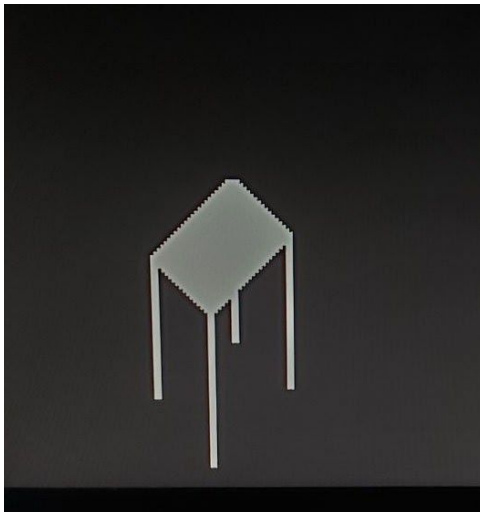
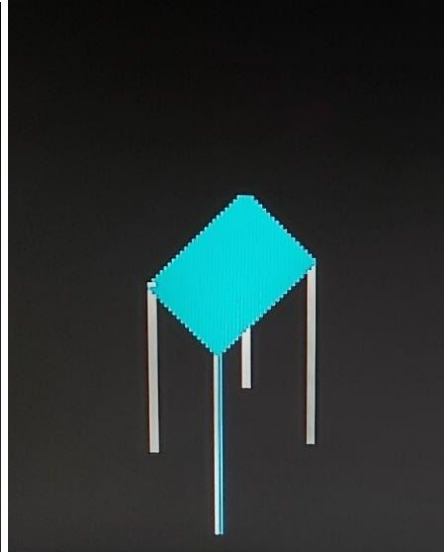
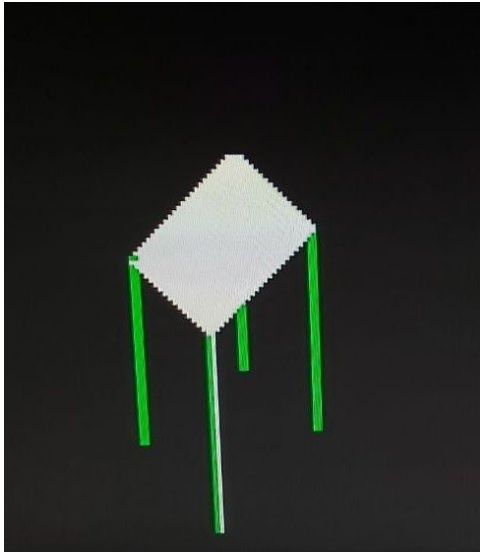
body @ 044 1dc 88 DRAWH
body @ 048 1E0 80 DRAWH
body @ 04C 1E4 78 DRAWH
body @ 050 1E8 70 DRAWH
body @ 054 1EC 68 DRAWH
body @ 058 1F0 60 DRAWH
body @ 05C 1F4 58 DRAWH
body @ 060 1F8 50 DRAWH
body @ 064 1FC 48 DRAWH
body @ 068 200 40 DRAWH
body @ 06C 204 38 DRAWH
body @ 070 208 30 DRAWH
body @ 074 20C 28 DRAWH
body @ 078 210 20 DRAWH
body @ 07C 214 18 DRAWH
body @ 080 218 10 DRAWH
body @ 084 21C 8 DRAWH

foot @ E0 1C8 0B0 DRAWV \ DESTRA VERTICALE
foot @ E4 1C4 0B4 DRAWV \ DESTRA VERTICALE
;
```

Il risultato, dopo aver cambiato le linee tramite il codice:

```
ff00ffa2 changeline2  
ff00ff00 changeline2  
line1 line2 table  
line3 line1 table  
ffffa10c changeline3
```

, è il seguente:



Il disegno in sè è molto semplice, ma permette di capire quali sono le potenzialità del framework, magari nell'ambito del disegno CAD (spiegato meglio nelle conclusioni).

5.1. Elenco Words

5.1.1. Disegno e Utilities

- DRAWXX¹ (LENGHT --)
- DRAWX² (color posx posy lenght --)
- OBB³ (LINE ANGOLO LENGTH --)
- Move⁴ (vedi note)
- TIMER (--)
- Clean (color --)
- Blankscreeny (--)
- Blankscreenn (--)
- Changeline (linenumber color --)
- Set/Get⁵ ()
- Setdefaultline (line --)
-

5.1.2. Esempi

- Table (colorbody colorfoot --)
- Screensaver1 (--)
- Screensaver2 (--)
- Stamp (number --)
- Heart (posx posy --)
- Hearts (--)
- Clean (--)

N.B I codici non commentati in questa sessione lo sono accuratamente nel file complete.f.

¹ DrawXX: è una famiglia di comandi, il nome del comando è in base alla direzione nell'asse y con Up oppure Down per effettuare una linea Verticale dal punto in cui è il cursore (DRAWUV, DRAWDV). Oppure se si vuole effettuare una linea orizzontale verso destra o sinistra dal punto in cui è il cursore i comandi sono (DRAWRH , DRAWLH)

² Draw: Sono due comandi: DRAWV e DRAWH, a differenza della famiglia DrawXX, queste words sono pensate per caricare immagini o disegni pre-configurati poiché nella prima famiglia bisogna lavorare con il cursore e con questa no.

³ Obb: è una famiglia di comandi, il nome del comando è in base alla direzione in cui si vuole fare la linea obliqua, ovvero UP/DOWN RIGHT/LEFT. e.s OBBUR OBBDL (Obliqua Up Righ, Obliqua Down Left)

⁴ Move: è una famiglia di comandi, il nome del comando è in base all'operazione che si vuole effettuare con il cursore: se si vuole spostare di un tot di numeri di pixel dalla posizione attuale allora i comandi sono: MOVEPX, MOVEPY (P sta per PLUS), se si vuole spostare completamente in una posizione dello schermo nell'asse xy, x oppure y allora i comandi sono rispettivamente (MOVEXY, MOVEX o MOVEY)

⁵ Sono state inseriti vari get e vari set delle proprietà della mailbox; è possibile, modificando i parametri, personalizzarli. <https://github.com/Mailbox-property-interface>

6. Future Implementazioni

(STAMPA NUMERI E CARATTERI)

Come possibile implementazione si è pensato alla possibilità di stampare a schermo dei caratteri (e numeri) pre-caricati all'interno di un array nel seguente modo:

1. Si crea un array con il/i carattere/i che si vogliono rappresentare
2. Si creano le combinazioni necessarie alla creazione del carattere seguiti da un numero identificativo particolare per concludere il ciclo di stampa (es. -1)
3. Si carica l'array nello stack
4. Si stampa con la funziona stamp

Un esempio di stampa possiamo vederlo con il numero 1 (è implementato anche il numero 0)

```
variable temp
( Carica nell'array specificato tutti i valori contenuti nello stack
: load ( -1 a1 a2 a3 ... an addr -- )
  0 i!
  temp !
  BEGIN
    temp @ i CELL !

    i+

    dup
    -1
    =
  UNTIL
  DROP
;

-1 1010101 1010101 1010101 10101010 10101010 10101010
0 10100 0 0 10100 0 0 10100 0 0 10100 0 0 10100 0 0
10100 0 0 10100 0 0 10100 101 0 10100 10101 0 10100
1010100 0 10101 1010000 0 10101 1000000 0 10101 0 0 1010101 0

num1 load
```

```

stampa ( a -- )

DUP
1
swap NumeroPalette swap CELL @
0 i!
BEGIN
  2DUP
  i cell @
  SWAP
  framebuffer i 3 mod 4 * + WIDTH i 3 / * +
  SWAP
  drop 30 2 WIDTH * * +
  EA 2 * +
  !

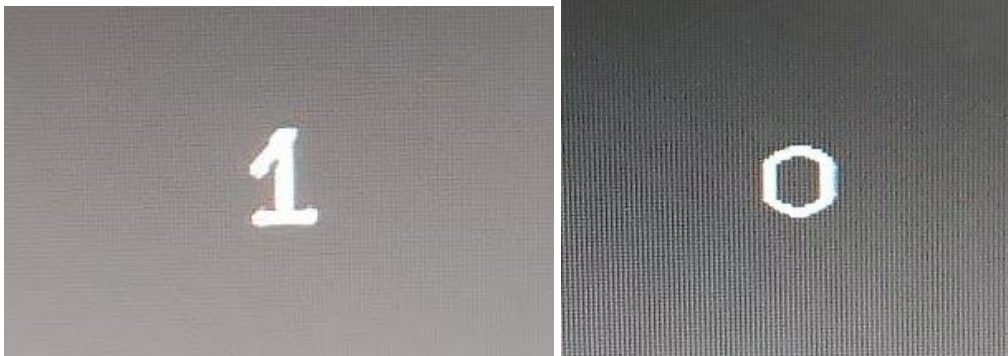
  i+
  i 50 =
UNTIL
DROP DROP DROP

```

utilizzando come sintassi:

1 STAMP oppure 0 STAMP

Il risultato è:



Oppure è possibile creare la propria scrittura tramite i comandi draw e obb, salvarli come words e stamparli all'occorrenza creando così la propria calligrafia.

N.B Il metodo adoperato si può considerare come " barbaro " in quanto molto pesante da realizzare sia dal punto di vista computazionale che dal punto di vista della programmazione (occorre fare pixel per pixel all'interno di un quadrato).

Invece utilizzando le funzioni di scrittura, ad esempio per fare 1, avremmo bisogno di:

- una linea obliqua alta verso destra di inclinazione 2 oppure 3.
- una linea verticale verso il basso di dimensione 6 oppure 8.
- spostamento del cursore di -12 x.
- linea orizzontale verso destra di 4 o 8.

Decisamente meno faticoso rispetto all'esempio 'barbaro' .

(Stampa di un'immagine passata da UART)

Come un'altra possibile implementazione si è pensato di far stampare un'immagine passata dal terminale, i vari problemi dell'operazione sono molteplici:

- Il tempo di esecuzione dell'operazione dato che l'UART può caricare fino a un 123 kbit/s.
<http://federico.defaveri.org/tag/ft232/>
- l'immagine dovrà essere creata in un modo particolare utilizzando il formato 8 bit.
- Oltre il formato inusuale dovrà avere come ultimi bit quelli creati per fermare i cicli su forth, ovvero -2 .

E' realizzabile ed abbastanza complesso ma un ottimo spunto per un progetto futuro.

Si potrebbe creare un dizionario di parole e all'occorrenza stamparle tramite la funzione (STAMP).

(Videogioco)

Utilizzando un sistema di input (tasti, breadboard, resistori) collegati alle due linee GPIO del raspberry è possibile creare un'interattività con l'utente e, con qualche "giochetto" grafico, possibile creare un videogioco.

Si pensa ad una versione più statica di tetris, un gioco di carte generico, un tic tac toe oppure un gioco ad indovinelli come " Chi vuole essere miliardario ? " .

Tutto graficamente realizzabile con i vari strumenti messi a disposizione in questo progetto ed una buona dose di buona volontà.

(Robot comandabile)

Utilizzando vari progetti creati da altri colleghi, sarebbe possibile creare un robot a guida differenziale con comandi dati da un'interfaccia grafica. Inoltre, implementando una connettività remota (come quella wireless, programmabile in forth) , sarebbe potenzialmente possibile creare un proprio robot con un budget veramente basso.

(Cursore interattivo)

Utilizzando un dispositivo esterno come tasti, è possibile muovere il cursore sullo schermo utilizzando l'accoppiata comando-word. E' realizzabile anche un cursore tramite mouse se si riuscisse a definire un pattern di comandi in base al movimento del dispositivo.

(Modellazione)

Nell'esempio del tavolo possiamo notare come è facile dividere le componenti dell'oggetto e colorarli a piacimento. E' possibile implementare un sistema CAD tramite questo framework modificando opportunamente le funzioni di draw (lasciate di proposito come esempi) scrivendo in funzioni a parte il disegno e richiamandole in seguito.

7. Conclusioni

7.1. Problemi

Il framework non è esente da errori, bisogna effettuare un rigoroso controllo poiché la posizione x nello schermo deve essere un multiplo di 4 (0 4 8 C in hex). Spostando difatti il cursore per pixel che non siano multipli di 4 e provando in seguito ad effettuare un DRAW verrà lanciata la seguente 'eccezione':

```
5 drawhr
Data abort exception accessing memory at address 0x1010101

00015060  E0 AA 00 00 58 84 00 00 04 00 00 00 84 99 8E A7 |....X.....|
00015070  6B CF AD D5 4D B3 B9 3E 01 01 01 01 00 40 09 00 |k...M..>.....@..|
00015080  10 17 02 00 68 90 01 00 44 85 00 00 DF 01 00 80 |....h...D.....|
00015090  A4 6A 64 6F BE 06 CE A2 04 EF E6 62 1F 85 AC 3B |.jdo.....b...;|
```

Per evitare questo errore si è pensato di moltiplicare la posizione 'x' per 4 in modo da fare diventare la posizione valida un multiplo di 4 e non avere errori, questo però provoca dei valori non corretti nello schermo.

Le soluzioni pensate per risolvere il problema verificatosi sono le seguenti:

- *Utilizzo della funzione modulo.* Questo tipo di soluzione dà però dei falsi valori di x (come la precedente).

- *Sistema di "arrotondamento" del numero.* Si prende il valore di x, si effettua il modulo 4 e nel caso il risultato di questa operazione sia diverso da 0 si modifica 'x':

Risultato modulo 1 -> 0

Risultato modulo 2 -> 4

Risultato modulo 3 -> 8

Anche questa soluzione adottata risulta però non soddisfacente a causa dell'arrotondamento stesso.

- *Azione di controllo.* Si è infine pensato di risolvere il problema attuando un controllo quando si effettua un movimento del mouse nell'asse x, di un valore non multiplo di quattro. Il risultato è accettabile ma non la ritengo la soluzione ideale.

Un'ulteriore problema manifestatosi è quello relativo al cursore. E' possibile settare il cursore in bianco (1010101), verde (2) oppure rosso (3) ma non in altri colori. Provando a cambiare il colore in uno diverso da quelli elencati , avremo degli artefatti in quanto il colore è impostato in 32bpp di default . Suppongo che il problema sia perché la profondità del colore settata è 8 nel framebuffer.

Soluzione provvisoria per i parametri x:

```
: MOVEPX
  DUP
  4 mod 0 =
  IF
    cursorx @ + cursorx !
  SETCURSOR
  ELSE
    ERROR
  THEN

;

: MOVEPY ...
;

: MOVEX
  DUP
  4 mod 0 =
  IF
    cursorx !
  SETCURSOR
  ELSE
    ERROR
  THEN

;
```

Si effettua un controllo sul parametro x, ovvero l'unico che da problemi, e si verifica che esso sia multiplo di 4.

Nel caso di riscontro positivo l'azione è consentita e lo spostamento viene effettuato, nel caso negativo viene mostrato un messaggio d'errore.

7.2. Commenti finali

Non essendo un progetto realizzato in Team, ma bensì ideato da solo, trovo il risultato abbastanza solido e soddisfacente. Ritengo inoltre possibili molte e mutevoli implementazioni e miglioramenti, gli stessi richiederebbero però l'inserimento del progetto di ulteriori membri. Difatti penso che uno scambio di idee e conoscenze sia necessario per migliorare un algoritmo e poter correggere eventuali errori coscientemente.

Spero, per coloro che si confronteranno con questo framework, che sia stato il più possibile chiaro in ogni passaggio.